Large Language Models are Few-shot Generators: Proposing Hybrid Prompt Algorithm To Generate Webshell Escape Samples

Mingrui Ma[1], Lansheng Han[1*], Chunjie Zhou[2]
[1] *Hubei Key Laboratory of Distributed System Security, Hubei Engineering Research Center on Big Data Security, School of Cyber Science and Engineering, Huazhong University of Science and Technology, Wuhan, China*
[2] *The Key Laboratory of Ministry of Education for Image Processing and Intelligent Control, School of Artificial Intelligence and Automation, Huazhong University of Science and Technology, Wuhan, China*
[1]{m202271767, 1998010309}@hust.edu.cn, [2]{cjiezhou}@hust.edu.cn

***Abstract*—The frequent occurrence of cyber-attacks has made webshell attacks and defense gradually become a research hotspot in the field of network security. However, the lack of publicly available benchmark datasets and the over-reliance on manually defined rules for webshell escape sample generation have slowed down the progress of research related to webshell escape sample generation strategies and artificial intelligence-based webshell detection algorithms. To address the drawbacks of weak webshell sample escape capabilities, the lack of webshell datasets with complex malicious features, and to promote the development of webshell detection technology, we propose the Hybrid Prompt algorithm for webshell escape sample generation with the help of large language models. As a prompt algorithm specifically developed for webshell sample generation, the Hybrid Prompt algorithm not only combines various prompt ideas including Chain of Thought, Tree of Thought, but also incorporates various components such as webshell hierarchical module and few-shot example to facilitate the LLM in learning and reasoning webshell escape strategies. Experimental results show that the Hybrid Prompt algorithm can work with multiple LLMs with excellent code reasoning ability to generate high-quality webshell samples with high Escape Rate (88.61% with GPT-4 model on VIRUSTOTAL detection engine) and Survival Rate (54.98% with GPT-4 model).**

***Index Terms*—webshell escape sample generation, Hybrid Prompt, large language models, prompt engineering**

I. INTRODUCTION

Webshell [1], as a typical example of malicious scripts exploiting injection vulnerabilities, allows hackers to remotely access and invade web servers, posing serious threats to socio-economic and network security. Webshells come in various forms, ranging from a single line of code that allows remote execution of user-provided system commands to large-scale complex script files. The code can also be written in multiple programming languages such as asp, aspx, php, jsp, pl, py, etc. Similar to the research on malware detection, webshell generation and detection are non-stationary, adversarial problems [2], which have been engaged in a constant game of cat and mouse, with an escalating spiral trend. From the attacker's perspective, mainstream webshell detection tools and engines like VIRUSTOTAL [3], WEBDIR+ and AVAST are frequently updated and maintained, incorporating the rules and characteristics of new webshells within days or even shorter periods. This forces attackers to constantly develop new webshell generation methods to bypass the detection of such engines. On the detection side, research is still in its infancy [4]. There is a lack of publicly available benchmark datasets and open-source baseline methods for webshell detection. Most models using neural networks or intelligent algorithms claim to have high accuracy and low false positives. However, the fact is that these models are basically tested on private datasets, which usually consist of only a few hundred or fewer samples, with obvious malicious features. Even the simplest multi-layer perceptron (MLP) structure can achieve high-precision detection on such datasets through overfitting. In a real cyber-attack environment, the authenticity and generalization ability of such methods are difficult to guarantee. For a limited number of publicly available webshell repositories on the internet, detection engines can also achieve high-precision detection (Figure 1), and the superiority of artificial intelligence (AI)-based methods is not fully demonstrated.

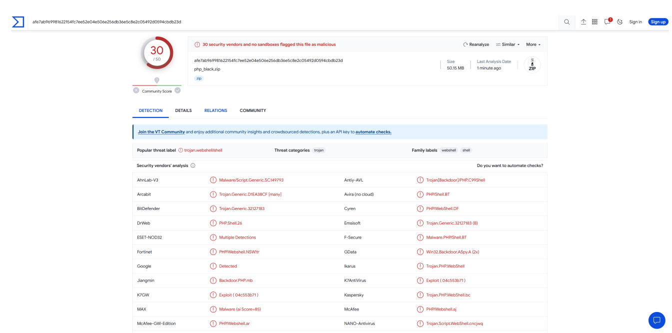

Fig. 1. VIRUSTOTAL achieves high precision detection on publicly available webshell repositories

In fact, Abdelhakim et al. [4] argued that AI methods excel at extracting abstract features in webshell, which are advanced features that go beyond lexical, syntactical, and semantical features. These advanced features help reveal hidden aspects in webshells that cannot be detected through syntax and semantic analysis. However, unlike the research on malware adversarial sample generation [2], [5]–[7], research on webshell escape sample generation is still a blank field, which is due to the fact that the existing webshell bypass strategies are numerous and complicated, and there is no specific systematic method to follow. Therefore, it is an urgent and highly significant work to propose a webshell escape sample generation algorithm and construct a corresponding webshell benchmark dataset.

On the other hand, the blooming development of large language model (LLM) and artificial intelligence generated content (AIGC) technologies [8] has already made an indelible impact in various domains such as chat and image generation [9]. As the latest achievement in the field of natural language processing (NLP), LLM has taken a significant lead over earlier neural network structures (i.e. LSTM [10], GRU [11], etc.) in contextual reasoning and semantic understanding capabilities. The widespread application of LLM in various code-related tasks (i.e. code generation [12], penetration testing [13], vulnerability detection [14], automated program repair [15], LLM fuzz tuning [16], vulnerability repair [17]) has fully showcased its excellent code reasoning abilities, making it possible to utilize LLM for generating webshell escape samples. Prompt engineering [18] plays a crucial role in the vertical research application of LLM, which aims to explore better ways of human interaction with LLMs to fully leverage their performance potential. Although the research related to prompt engineering is controversial, scholars opposing it argue that prompt engineering is too "mystical" and exacerbates the lack of interpretability of neural network models. However, it is undeniable that many key techniques in prompt engineering, such as Chain of Thoughts (CoT) [19], Tree of Thoughts (ToT) [20], Zero-Shot CoT [21], etc., have improved the reasoning abilities of LLMs. In addition, the application of techniques like LAnguage Model Analysis (LAMA) probes [22] have been gradually enhancing the interpretability of the models. Novel studies in prompt engineering, such as prompt finetuning, have been able to fine-tune the parameters in LLM, thus simplifying the traditional fine-tune process [23]. Moreover, AIGC technology is so "creative", that just a simple prompt can make LLM produce a 0-Day webshell (Figure 2).

Therefore, in this work, we explore the unexplored research area of AIGC-enabled webshell escape sample generation strategies. We propose Hybrid Prompt, a hierarchical and modular prompt generation algorithm, and apply it to different LLM models to generate multiple webshell samples with high escape capabilities. Experimental results demonstrate that the escape samples generated by the Hybrid Prompt algorithm + LLM model can bypass detection by mainstream detection engines with high Escape Rate (ER) and Survival Rate (SR).

The main contributions of this paper are summarized as follows:

- We propose Hybrid Prompt algorithm, which combines the advantages of multiple prompt schemes such as ToT [20], few-shot prompting, CoT, etc. By synthesizing key features related to webshell escape and designing prompt strategies tailored to different sizes of webshells, the algorithm effectively enhances the code reasoning ability of LLM models and generates high-quality webshell escape samples.
- We construct a webshell benchmark dataset generated by the Hybrid Prompt algorithm. This dataset achieves high ER and SR among mainstream detection engines and reflects the performance of rule-based detection engines more realistically and effectively. It has a high reference value to the research in the field of webshell detection.
- We investigate and compare the quality of escape samples generated by different LLM models using the Hybrid Prompt algorithm. All these samples exhibit high ER, surpassing webshell samples generated by other intelligent algorithms (e.g. genetic algorithm [24]).
- Our study confirms the possibility of applying LLM techniques in the webshell domain and inspires future research in the field of LLM-based webshell detection.

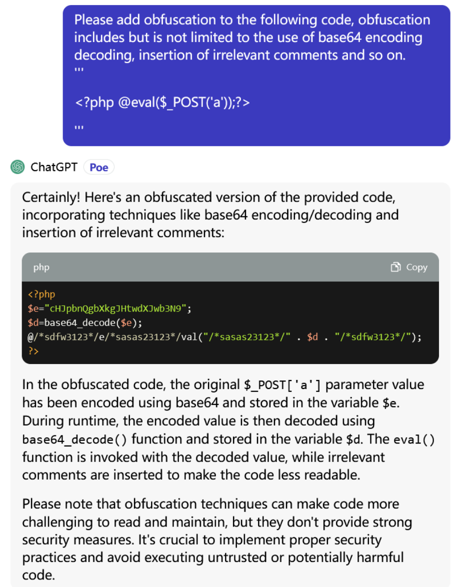

Fig. 2. Generating webshells using simple prompts on LLM

We believe that the incorporation of LLM and prompt engineering techniques can inject fresh blood into the research of webshell generation and detection.

II. PRELIMINARY

With the development of AIGC and LLM technologies, there are numerous LLM models in different subfields with different focuses. For example, GLM [25] and GLM2 models tend to prioritize open-source and lightweight to meet the deployment needs of personal terminals. DALLE [26] focuses on AI image generation, while FATE-LLM [27] is biased towards application scenarios under the federal learning paradigm. Hybrid Prompt performs exceptionally well on LLM models with strong code reasoning abilities. We have tried to apply basic prompts on Chatglm-6B, Chatglm2-6B, Chatglm-13B, and Chatglm2-13B models, but the performance is unsatisfactory, as shown in Figure 3.

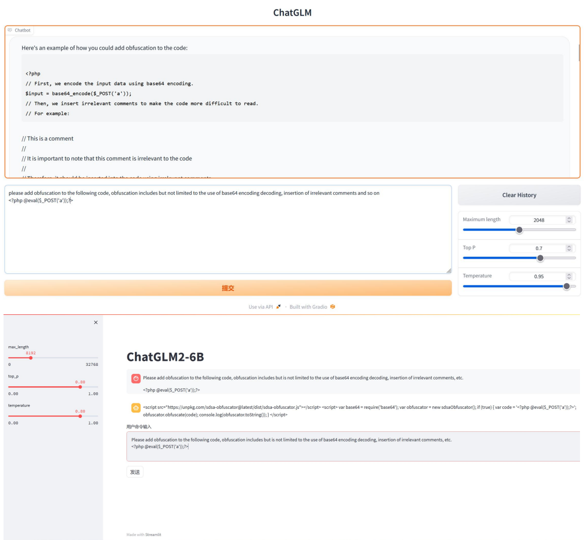

Fig. 3. Chatglm and Chatglm2 models perform poorly on the task of webshell generation

Even when adjusting key parameters such as $Temperature$, Top_p, Top_k, etc., or even fine-tuning such models, the results remain unsatisfactory. The fundamental reasons are twofold. Firstly, Chatglm and other LLM models focusing on interactive dialogs have weak reasoning ability, while the webshell escape sample generation task requires strong inference ability. (The model should effectively understand each specific escape strategy in the prompts and modify the given examples for bypassing without destroying the original functionality and syntactic structure of the webshell.) Secondly, prompt engineering itself tends to have more significant effects on LLMs with more than 10B parameters. Therefore, this strategy is more suitable for LLM models with a large number of parameters and strong code reasoning abilities, such as GPT-3.5 and GPT-4.

III. ALGORITHM DESIGN

A. Overall Workflow

The overall flow from collecting multi-source webshell scripts to generating webshell escape samples is shown in Figure 4.

The principle and operation process of each part will be expanded in detail later.

B. Data Filtering

To facilitate the implementation of the Hybrid Prompt algorithm, we need to construct the Template webshell dataset. Since webshell scripts collected from multiple sources are diverse in types and have confusing names (i.e. China Chopper, b374k, etc.), and the Template webshell dataset requires clean and well-characterized webshell scripts, we perform triple data filtering process on Multi-source webshell scripts, as shown in Figure 5.

In the first filtering step, we calculate the *MD5 hash* value of all scripts to filter out webshell scripts with consistent content but confusing names. The filtered scripts are then renamed using their corresponding hash values. In the second filtering step, we convert the webshell scripts into Abstract Syntax Tree (AST) structures to filter out the scripts with the same syntax structure. For php scripts, we use "*php-ast*" to perform the translation (*ast\parse_code*) and add the *nameKind* attribute to the nodes. We process the child nodes belonging to the array and AST separately. The pseudocode and example for this step are shown in Algorithm 1 and Figure 6.

In the third filtering step, the Vulcan Logic Disassembler (VLD) module in Zend disassembler is used to disassemble the scripts into opcode structures, aiming to filter out webshell scripts with consistent execution sequences. A typical example is shown in Figure 7.

C. Hybrid Prompt

The ToT method has significant performance advantages over CoT, Self Consistency (SC) [28] method in solving complex reasoning problems by searching for multiple solution paths, using strategies such as backtracking and pruning, similar to human thinking rather than the traditional auto-regressive mechanism of making token level decisions one by one in a left-to-right manner. This allows it to better handle heuristic problems like genuine problems. Therefore, we also leverage and innovate this process paradigm in the complex reasoning task of webshell escape sample generation. The overall flowchart of the Hybrid Prompt algorithm is illustrated in Figure 8.

Before proceeding, let's first formalize some relevant symbols. We use M to denote LLM, o to denote one of the candidates generated by each thought of Hybrid Prompt, O to denote the set composed of candidates, x to denote the original input of Hybrid Prompt, F_e to denote the few-shot example, N to denote the tree depth of Hybrid Prompt and p to denote the number of candidates.

1) Thought Decomposition:

ToT argues that a suitable thought should be able to generate promising and diverse samples, facilitating LLM in assessing its problem-solving prospects. However, compared to tasks with clear rules such as *Game of 24*, *5*5 Crosswords*, etc., the thought search space for webshell escape sample generation is broader and more challenging.

To address this, we have developed a webshell escape sample generation whitepaper by taking into account the characteristics of webshell escape samples. The whitepaper consists of numerous escape keywords representing different escape methods. We refer to each keyword as a module, and some modules further have secondary and tertiary modules. This hierarchical structure of modules constitutes a forest structure, in which each primary module is the root node of the tree in the forest. This modular design concept has strong scalability, allowing for the real-time addition of modules to increase the number of escape methods for the Hybrid Prompt algorithm, as shown in Figure 9.

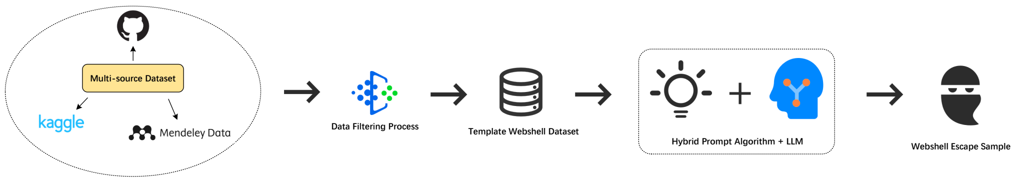

Fig. 4. The overall workflow of webshell escape sample generation

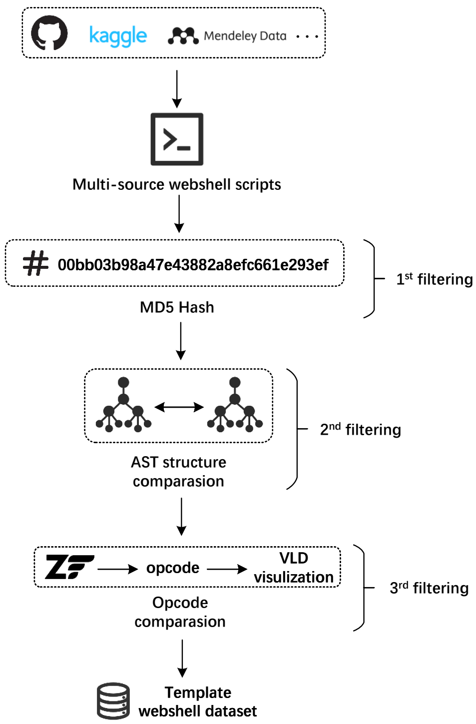

Fig. 5. Triple data filtering process

Algorithm 1 Php-ast Runtime Flow

```
$ast = ast\parse_code($code, $version=70);
$new_ast = add_attr($ast);
$json = json_encode($new_ast, JSON_PRETTY_PRINT
  | JSON_UNESCAPED_UNICODE | JSON_OBJECT_AS
  _ARRAY);
```

Therefore, in Hybrid Prompt, thought is set as the search space for LLM contemplating Template webshells based on a module.

2) Thought Generator $G(M, o)$:

Since webshell escape sample generation is a heuristic problem, we apply the CoT method to each module for generating multiple intermediate webshell samples. Considering that LLM may generate some low-value solutions with large deviations from the expectation, thus reducing the efficiency of subsequent votes, we design F_e chain structure for each module, as shown in Figure 10.

Therefore, $G(M, o) = M(F_e, o)$.

Each node in the F_e chain includes the original webshell sample, as well as the webshell sample processed by the corresponding module, and a brief description explaining the processing method and core ideas of the module. When filtering the F_e chain, we follow the following principles:

- The structure of the example webshell code should be as simple as possible.
- Each node contains, as far as possible, only the processing methods corresponding to that module.

The purpose is to reduce the difficulty of LLM in learning the corresponding method through an example that is as simple as possible and contains the core idea. The descriptive explanation further enhances the interpretability of the solutions. This idea is also in line with the logical process of human learning and cognition, e.g., "from shallow to deep," to help LLM better learn the features of the methods.

F_e can essentially "modify" the LLM's thinking direction to a certain extent so that the LLM can be generated in a Few-shot CoT mindset. In most cases, each F_e chain contains multiple F_e examples to provide more comprehensive coverage of different scenarios. In this case, multiple nodes are used as input prompt components for the current iteration round, to help LLM better learn multiple segmented strategies. Due to the large search space and sample diversity for each module, this Few-shot CoT method yields better results.

Meanwhile, based on the input webshell size, we design 2 different generation approaches. For small webshells, we include p candidate webshell samples in a single conversation returned by the LLM. In this case, the average maximum length of each candidate webshell sample $L(Avg\_Candidate_i)$ is calculated as (1):

$$L(Avg\_Candidate_i) = (L(MaxToken) - L(InputPrompt))/p. \quad (1)$$

Where $L(MaxToken)$ denotes the maximum context length that the current LLM model can receive, and $L(InputPrompt)$ denotes the length of the input prompt in the current thought. Since small webshells are generally shorter, this approach can save the consumption of LLM's token resources, and enable LLM to generate more diverse samples in the returned message of a single conversation through specific "key prompts".

For large webshells, we enable the n parameter function to generate p candidate webshell samples by receiving multiple return messages from LLM. In this case, the maximum

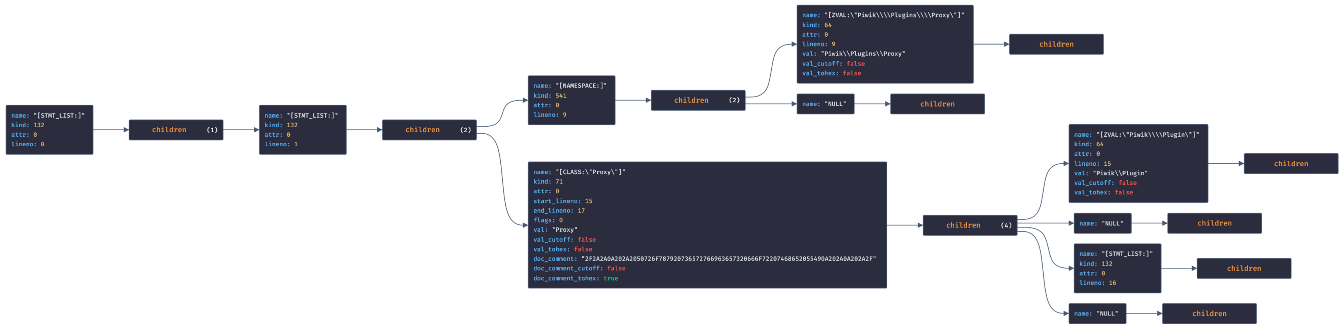

Fig. 6. An example AST structure in the second filter step

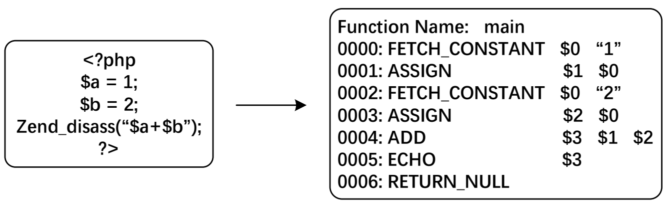

Fig. 7. A typical example of generating opcodes through VLD disassembler

length of each candidate webshell sample $L(Candidate_i)$ is calculated as (2):

$$L(Candidate_i) = L(MaxToken) - L(InputPrompt) - L(Description_i) \tag{2}$$

Where $Description_i$ represents the brief description generated by LLM for the i^{th} candidate webshell sample, which is used to summarize the idea of candidate webshell generation and facilitate the subsequent voting process. This approach maximizes the length of the generated candidate webshell sample at the expense of consuming more token resources.

3) State Evaluator $V(M, O)$:

Corresponding to Thought Generator, State Evaluator is also designed to have 2 different voting methods for large and small webshells. For small webshells, Hybrid Prompt uses LLM to vote on multiple intermediate webshell samples (states) and filter out the optimal ones. The reason for voting on multiple samples instead of voting on solutions is as follows:

- Since the Thought Generator operates in a few-shot CoT mindset, webshell samples help LLM evaluate and assess the differences between generated examples more intuitively to make optimal judgments.
- Voting directly on the samples can preserve all the original information of the candidate webshells.

In this case, $L(Generator(Input + Output)) \approx L(Evaluator(Input + Output)) < L(MaxToken)$. Because both contain F_e, the webshell contents of p candidates, and additional prompt information.

For large webshells, it is not feasible to directly input the webshell contents of p candidates into LLM because $p \times (L(Candidate_i)) + L(F_e chain) + L(Additional_{Prompt}) > L(MaxToken)$. Therefore, we use $Description_i$ instead of $Candidate_i$ as the input component of the vote procedure. This kind of information compression idea will inevitably lose the original code information. Figure 11 shows the 2 different vote ideas more intuitively.

Regardless of the vote idea, for $V(M, O)$, where $O = \{o_1, o_2, ..., o_p\}$, $V(M, o_i) = 1$ is considered a good state, when $o_i \sim M^{vote}(o_i|O)$. For Hybrid Prompt, the evaluation of a good state is to synthesize both the confusion level of the intermediate results generated by LLM for a module and the distance between them and the F_es. By allowing LLM to pursue local optimal solutions at each step of sample generation, this "greedy" idea makes it easier for the LLM to approximate the global optimal solution for the heuristic problem of escape sample generation.

4) Search Algorithm:

For the Hybrid Prompt method, the depth of the tree N corresponds to the number of modules (including the complete module structure such as primary and secondary modules). The DFS strategy leads to an excessive state space of LLM during the backtracking and pruning stages, which reduces the efficiency of the algorithm operation. Therefore, we consider using the BFS search algorithm. The pseudocode of the corresponding Hybrid Prompt-BFS algorithm is shown in Algorithm 2.

Taking into account performance and efficiency considerations, for the escape sample generation task, we set the number of candidates p to 1. The final output of the webshell escape sample is the candidate that wins in the vote process at the N^{th} layer.

5) Contextual Memory Range:

Since LLM has a limited range of contextual memory, we cannot let LLM memorize the entire Hybrid Prompt context but should set its local memory range. For Hybrid Prompt itself, it is impossible to compress the history information like many NLP tasks (e.g. contextual conversations) because it would result in a significant loss of raw webshell information. For this reason, our approach is to set the Contextual Memory Range for the Hybrid Prompt, as shown in Figure 8. Contextual Memory Range refers to the scope of each iteration in the Hybrid Prompt algorithm. At this stage, the only contextual information required for the next iteration round is the candidate

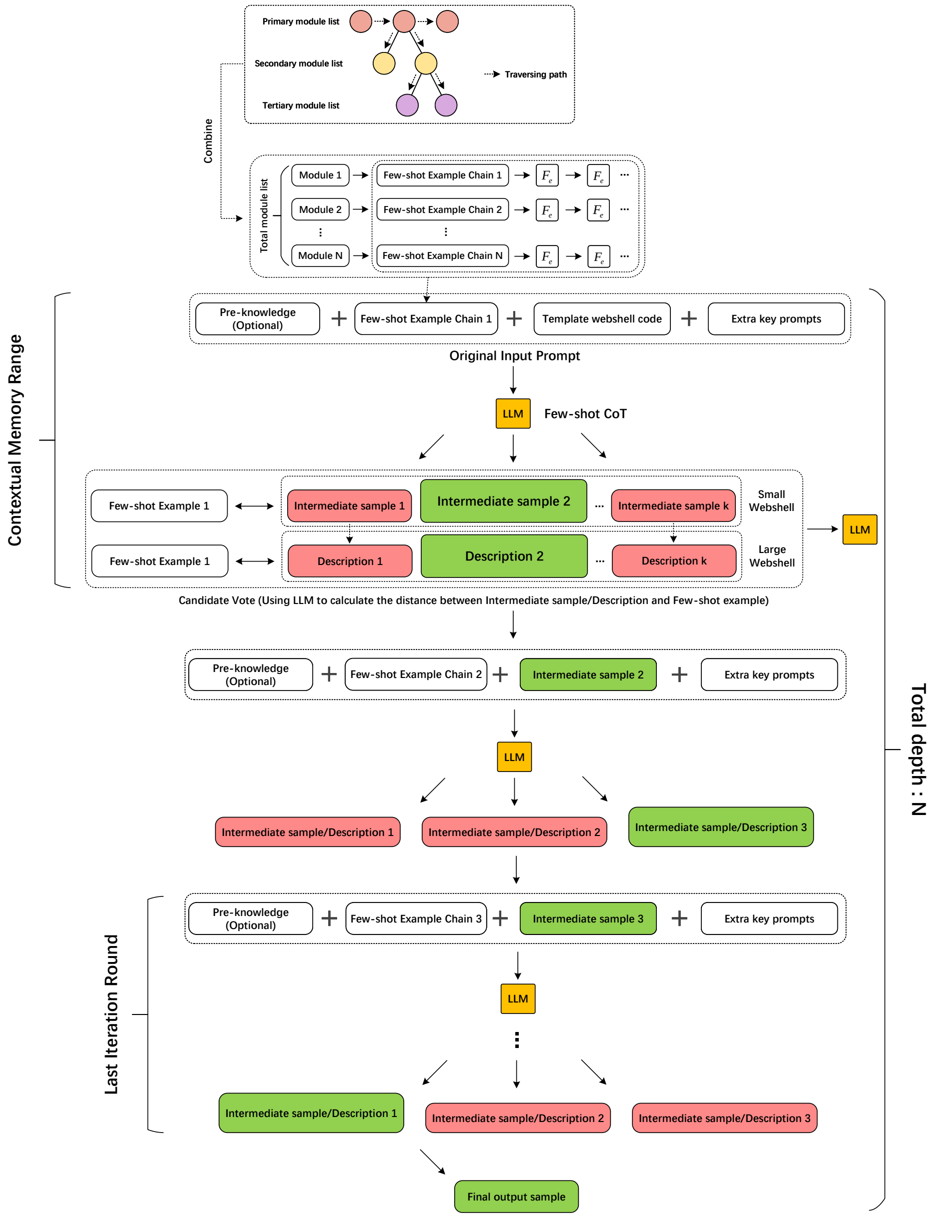

Fig. 8. The flowchart of Hybrid Prompt algorithm

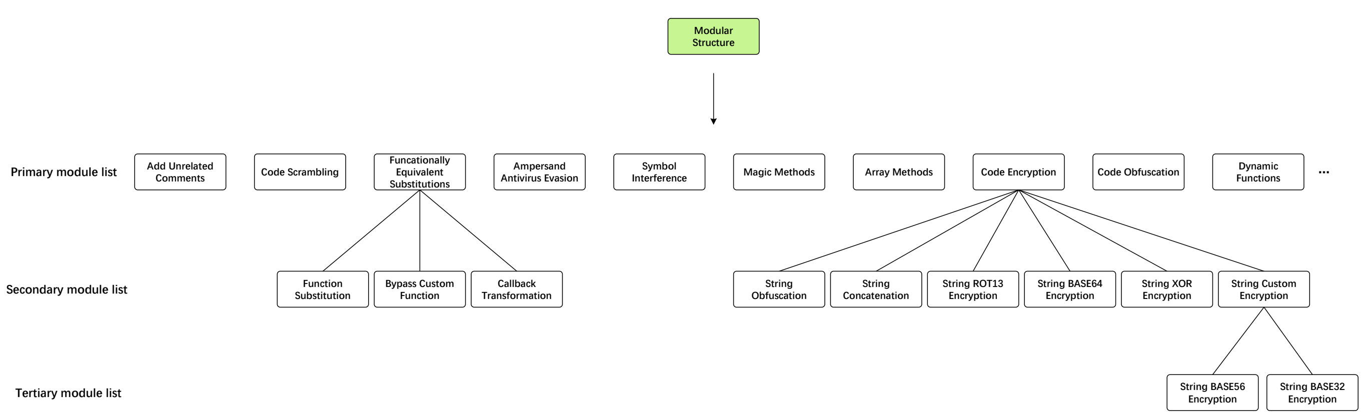

Fig. 9. Hierarchial module structure in Hybrid Prompt

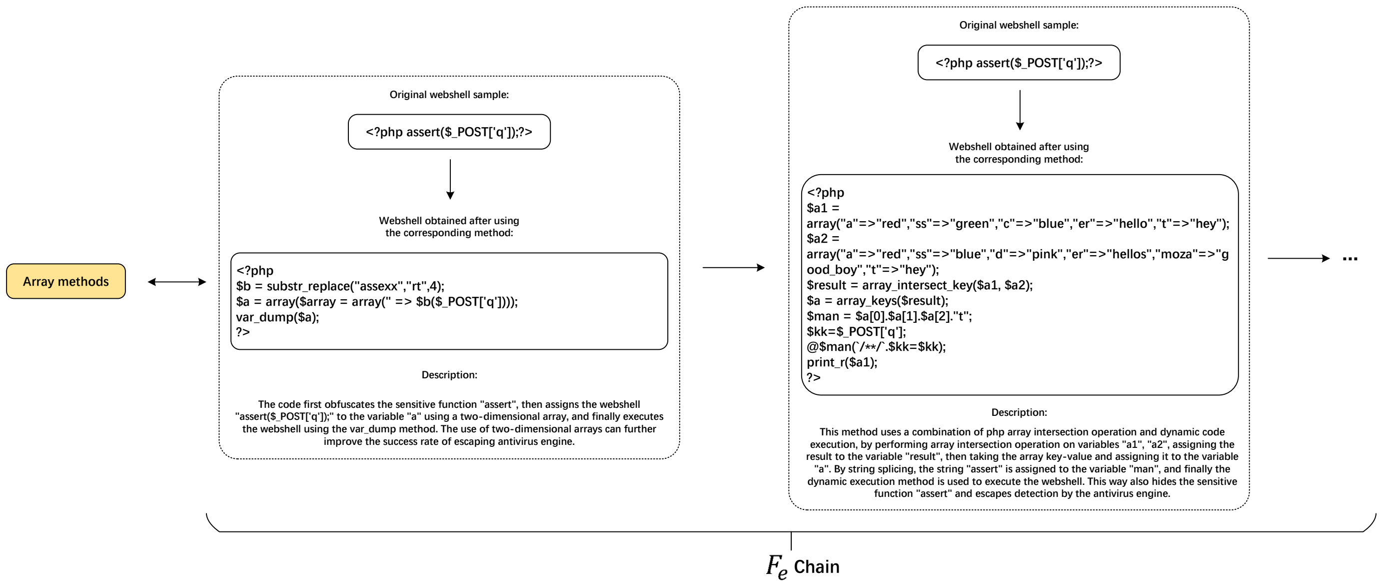

Fig. 10. The structure of F_e chain for each module

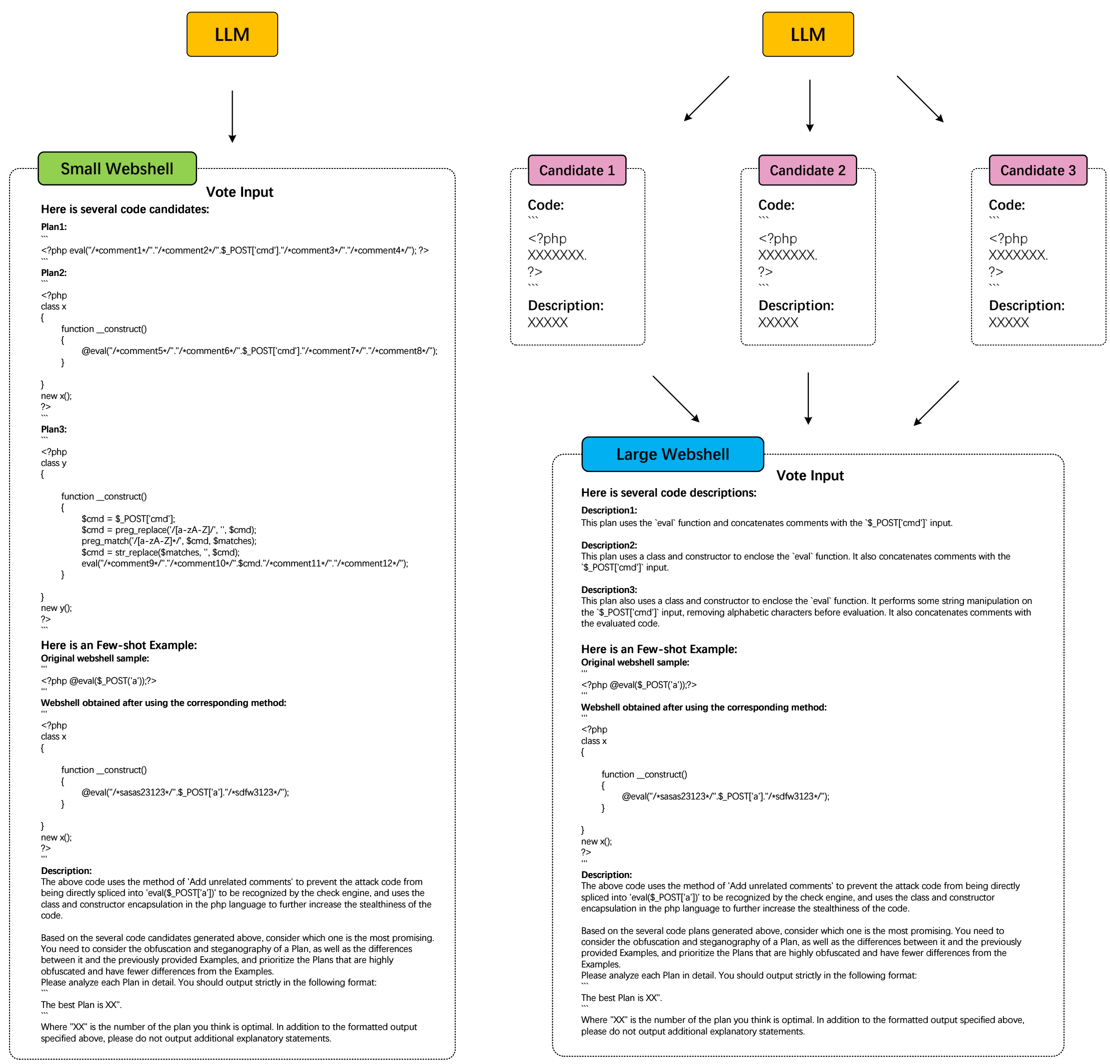

Fig. 11. Comparison of 2 different vote ideas

Algorithm 2 Hybrid Prompt-BFS Algorithm

Require: Input x, Thought Generator $G(M, o)$, State Evaluator $V(M, O)$, Tree Depth N, Candidate num p, Step Output $O_i (O \leq i \leq N)$

$O_0 = x$
for n = 1 to N **do**
  $O_n^{'} = \{[o, z] | o \in O_{n-1}, z_n \in G(M, o)\}$
  $V_n = V(M, O_n^{'})$
  $O_n = sort(V_n, p)$
end for
Return O_n

output selected by the winning vote strategy in the previous iteration. Therefore, defining the Contextual Memory Range ensures the continuity of information memory throughout the complete Hybrid Prompt algorithm. Correspondingly, $O_n^{'}$, V_n within the body of the "for" loop in the pseudocode are the local contextual contents that LLM needs to memorize.

6) Additional Explanation:

For the webshell escape sample generation task, an important guiding principle is to ensure the validity of generated samples. This means that the escaped samples should not lose the attack behavior and malicious features of the original samples and can be executed correctly without any syntax or lexical errors. To achieve this, Hybrid Prompt introduces Safeguard Prompt to constrain sample generation and improve SR. In addition, common techniques in prompt engineering, such as "‘ delimiter, are also applied in the Hybrid Prompt algorithm to normalize the output of LLMs.

The order of modules also has a significant impact on the Hybrid Prompt algorithm, as shown in Figure 12.

In Figure 12, if the "String XOR Encryption" module is placed in front of the "Symbol Interference" module, the encrypted webshell sample is no longer "text-readable", resulting in a high probability of hallucination when LLM executes to the "Symbol Interference" module, and triggering a series of subsequent generation errors. Therefore, when running the Hybrid Prompt algorithm, it is important to consider the relative position between specific modules and establish corresponding rules to avoid such situations from occurring.

IV. EXPERIMENTS

A. Setup

In the experimental section, our main objective is to answer the following questions:

RQ1: Can LLM effectively generate escape samples, and what is the ER of these samples under different detection engines?

RQ2: Are the individual parts of the Hybrid Prompt algorithm effectively designed?

RQ3: Does the number of candidates p have any effect on the performance of the Hybrid Prompt algorithm?

Experimental Environment. We first introduce the specific experimental environment, as shown in Table I.

TABLE I
EXPERIMENTAL ENVIRONMENT

CPU	Intel Xeon(R) Gold 6326 CPU @ 2.9GHz
RAM	64GB
GPU	NVIDIA TESLA A100-SXM4-80G ×2
Language	Python 3.10+
AI Framework	PyTorch 1.8.1+
Virtual Attack Environment	DVWA + AntSword

Particularly, we use a virtual environment simulating a vulnerable server in DVWA and apply AntSword virtual environment for attack testing, which is shown in Figure 13.

Evaluation Metrics. To better compare the quality of samples generated by different LLM models using the Hybrid Prompt algorithm, we choose two evaluation metrics: ER and SR, which are calculated as follows:

$$ER = 1 - DR = 1 - N_{Detected\_samples} / N_{Total\_samples} \quad (3)$$

$$SR = N_{Malicious\_samples} / N_{Total\_samples} \quad (4)$$

Where Detection Rate (DR) represents the detection accuracy of the detection engine, $N_{Total\_samples}$ is the total number of samples generated by LLM under the Hybrid Prompt algorithm, $N_{Detected\_samples}$ is the number of webshells successfully identified by the detection engine, and $N_{Malicious\_samples}$ is the number of samples generated by LLM under the Hybrid Prompt algorithm that still retain malicious functionality.

Models & Detection Engines. We test the ER and SR of samples generated by Hybrid Prompt under three detection engines: Web Shell Detector, WEBDIR+ and VIRUSTOTAL respectively. By calling the VIRUSTOTAL scanning API, we test a total of more than 58 different detection engines (i.e. AVG, ClamAV, AVAST, etc.). When any of these detection engines recognize the current test sample as a webshell, we consider that the sample fails to escape from VIRUSTOTAL. In addition, we cross-check the performance of several LLM models, including GPT-3.5, GPT-4, and Code-llama-34B, which demonstrate excellent performance in code generation and semantic understanding tasks.

Comparative Methods. Due to the lack of relevant research in the field of webshell escape sample generation, we also include a comparison with the dataset from CWSOGG [24], an obfuscated webshell dataset generated using the genetic algorithm.

Additional Explanations. By default, we set the number of candidates p to 3. It should be noted that to correctly calculate the SR of the samples, we set all the test sets to be labeled as "black" (regardless of whether they are truly malicious or not). Due to the frequent updating and maintenance of detection engines, the actual test results may differ slightly from the results presented in this paper. However, the experimental

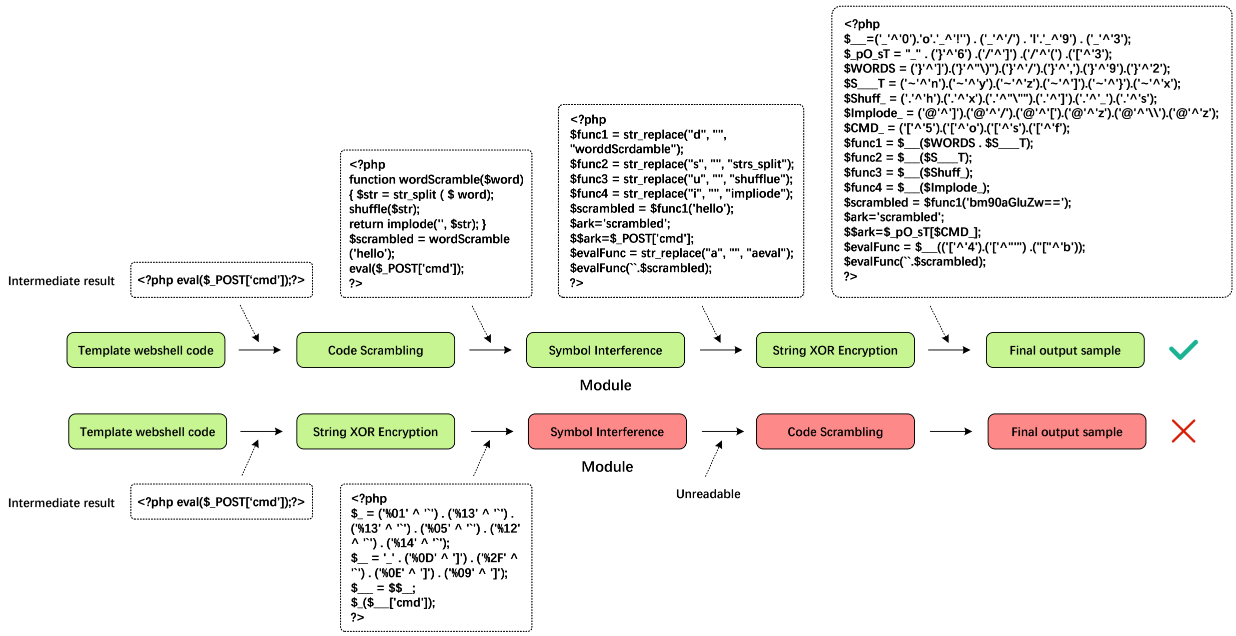

Fig. 12. Effect of module order on the Hybrid Prompt algorithm (Incorrect module order can result in abnormal output from the LLM)

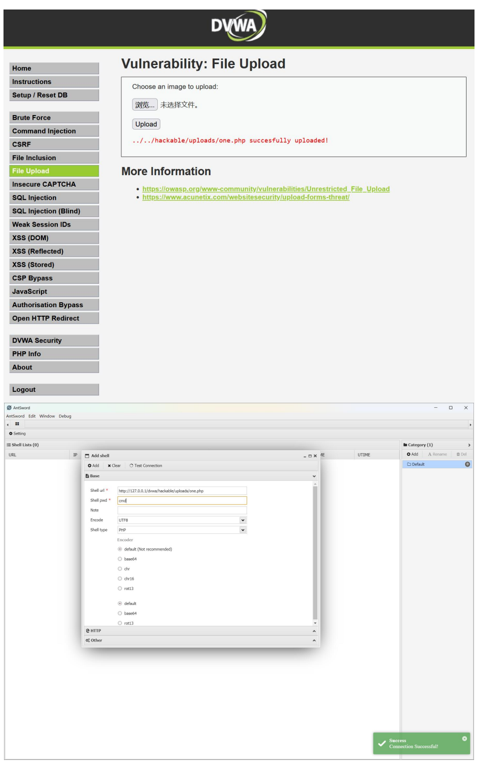

Fig. 13. Virtual attack environment

results in this paper can still effectively reflect the performance differences and data trends among different methods.

B. Comparative Experiment

All LLMs are generated under the complete Hybrid Prompt algorithm to achieve optimal performance. To answer **RQ1**, the comparative results are shown in Table II.

As can be seen in Table II, the GPT-4 + Hybrid Prompt algorithm has the best comprehensive performance, leading to both ER and SR. This is due to the fact that GPT-4 is more capable of following complex instructions carefully, while Hybrid Prompt contains multiple detailed instructions with normalized constraints. GPT-3.5, on the other hand, has a higher probability of partially following complex instructions, resulting in a higher probability of generating escape samples that prioritize either ER or SR, making it difficult to balance both. It is encouraging to note that the comprehensive performance of the open-source fine-tuned LLM Code-llama-34B, is very close to that of the GPT-3.5 model, confirming the performance potential of the open-source models. Meanwhile, we note that the ER of webshell samples generated by the 3 LLM models + Hybrid Prompt algorithm have far exceeded those of the Original Template Dataset and the CWSOGG Dataset, which fully demonstrate the performance superiority and dominance of the LLM models over rule-based artificial escape strategies and the traditional intelligent algorithms (i.e., genetic algorithm). As for the detection engines, VIRUSTOTAL, due to its integration of dozens of different detection engines, has a higher overall DR compared to Web Shell Detector and WEBDIR+. However, even VIRUSTOTAL struggles with the creativity of LLMs and the uncertainty of the generated escape samples, which illustrates the limitations and drawbacks of this type of specific rule-based detection engines.

Figure 14 visualizes the performance differences as reflected in Table II intuitively.

C. Ablation Analysis

To further validate the effectiveness of the Hybrid Prompt algorithm and address **RQ2**, we test the performance of sample

TABLE II
COMPARATIVE EXPERIMENT RESULTS

Anti-Virus Engine Model	***Web Shell Detector***	***WEBDIR+***	***VIRUSTOTAL***	SR
	ER			
GPT-3.5 Turbo + Hybrid Prompt	0.9342	0.8874	0.7465	0.4093
GPT-4 + Hybrid Prompt	0.9727	0.9287	0.8861	0.5498
Code-llama-34B + Hybrid Prompt	0.9015	0.8549	0.6358	0.3021
Original Template Dataset	0.3415	0.2054	0.1232	1
CWSOGG Dataset	0.4052	0.3151	0.2327	1

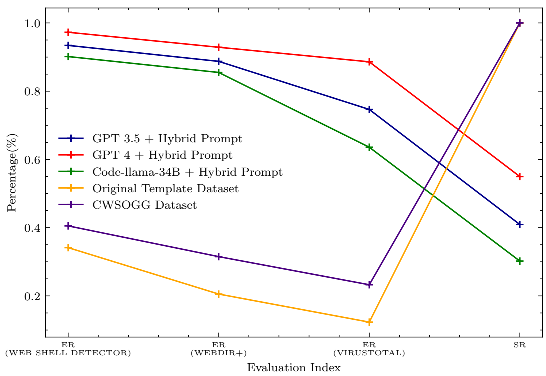

Fig. 14. Performance comparison of different LLM models on Hybrid Prompt algorithm

sets generated by removing different components of Hybrid Prompt under ER and SR evaluation metrics in the GPT-3.5 model. Specifically, we refer to the complete Hybrid Prompt algorithm as Strategy 1, removing the Safeguard Prompt as Strategy 2, removing F_e chain as Strategy 3, removing the vote strategy and generating only 1 sample per module as Strategy 4, and not using the Hybrid Prompt algorithm at all and letting the LLM directly generate webshell as Strategy 5. The experimental results are shown in Table III.

From Table III, it can be seen that Strategy 5 has poor performance and a high probability of hallucination due to the absence of any additional prompt. Both Strategy 3 and Strategy 4 produce different degrees of performance degradation. For Strategy 3, LLM loses reference examples, leading to a higher probability of generating corrupted samples. Strategy 3 also indirectly reflects that the current LLM's code reasoning ability still relies on F_e chains to achieve better task performance. For Strategy 4, LLM is unable to explore multiple reasoning paths, so the generation space and diversity of samples are limited, which leads to a lower ER. Strategy 2 has the least impact on the quality of generated escape samples. Although the probability of generating corrupted samples increases and the SR decreases due to the loss of Safeguard Prompt's normalization measures, the impact on the ER is not significant. However, collectively, for Strategy 2 - Strategy 5, all produce varying degrees of performance degradation compared to the complete Hybrid Prompt algorithm, fully demonstrating the effectiveness of various components of the Hybrid Prompt algorithm.

Figure 15 visualizes the performance differences as reflected in Table III intuitively.

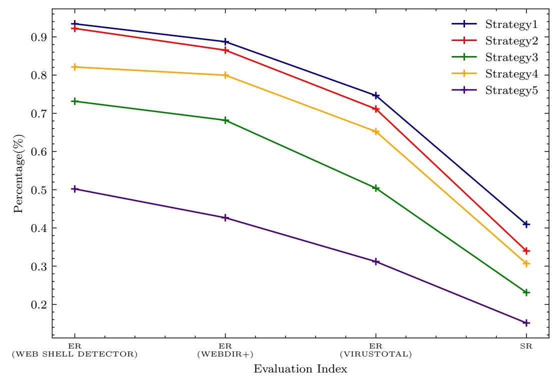

Fig. 15. Visualization of ablation analysis results for Hybrid Prompt algorithm

D. Sensitivity Analysis

We investigate the impact of the candidate number, p, on the SR and ER evaluation metrics of generated samples in the GPT-3.5 model. The experimental results of **RQ3** are shown in Table IV.

From Table IV, it can be observed that a larger number of candidates can increase the search space of LLM, which in turn enriches the diversity of generated samples, enables better selection of the optimal solution, and improves the sample ER and SR. However, the increase of p will also result in a higher token consumption and, in the case of small webshells, further reduces the $L(Avg\_Candidate_i)$ for each sample. Figure 16 is able to visualize the "marginal effect" that occurs as p increases (The yellow and purple folds in Figure 16 almost overlap). When p exceeds 3, the performance improvement of ER and SR metrics is not obvious, which can be attributed to the fact that the search space of LLM's self-inference is approaching the local upper limit. However, it is noteworthy that the consumption of tokens exhibits an almost linear relationship with the increase in p, despite the limited performance gains in ER and SR metrics. Therefore, the pros and cons between evaluation metrics and resource consumption should be weighed in practical applications.

V. DISCUSSIONS & LIMITATIONS

1) The Hybrid Prompt algorithm currently supports a limited number of webshell languages (initially only sup-

TABLE III
THE COMPARATIVE RESULTS OF ABLATION ANALYSIS

Anti-Virus Engine / Strategy	*Web Shell Detector*	*WEBDIR+*	*VIRUSTOTAL*	SR
	ER			
Strategy 1	0.9342	0.8874	0.7465	0.4093
Strategy 2	0.9221	0.8653	0.7114	0.3398
Strategy 3	0.7315	0.6819	0.5042	0.2310
Strategy 4	0.8213	0.7998	0.6524	0.3067
Strategy 5	0.5021	0.4267	0.3120	0.1513

TABLE IV
THE COMPARATIVE RESULTS OF SENSITIVITY ANALYSIS

Anti-Virus Engine / Candidate num p	*Web Shell Detector*	*WEBDIR+*	*VIRUSTOTAL*	SR
	ER			
1	0.8213	0.7998	0.6524	0.3067
2	0.8749	0.8567	0.7031	0.3648
3	0.9342	0.8874	0.7465	0.4093
4	0.9489	0.8968	0.7621	0.4163
5	0.9522	0.9014	0.7708	0.4266

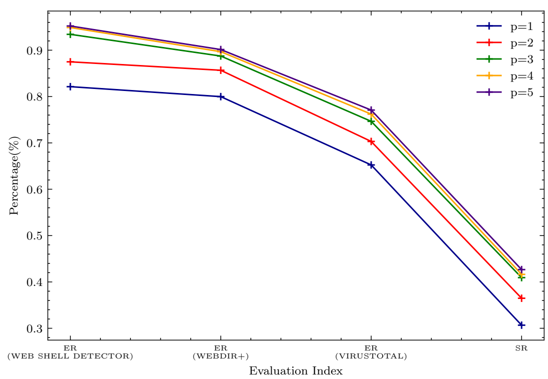

Fig. 16. Visualization of the "marginal effect" with increasing p

ports php language), and there is a need to expand it to support more webshell languages in the future.

2) Hybrid Prompt algorithm does not fine-tune LLMs. Fine-tuning can further reduce the probability of LLM hallucination and improve the quality of generated escape samples.
3) For the vote strategy in the case of large webshells, the description-based strategy used in the Hybrid Prompt algorithm results in the loss of original information from candidate code, which in turn affects the vote effect of LLM. While information compression strategies are acceptable for NLP tasks such as contextual dialogs, there is room for further improvement for tasks such as code generation, which require precise raw sample information.

VI. RELATED WORK

Due to the lack of relevant research in the field of webshell escape sample generation, we survey the technology domains associated with the Hybrid Prompt algorithm and the webshell detection task.

A. *Prompt Engineering Algorithm.*

As one of the most classic prompt algorithms, CoT [19] aims to assist LLMs in achieving complex reasoning abilities through intermediate inference steps. Zero-shot CoT [21], as a follow-up to CoT, enables LLM to perform self-reasoning through twice generation, involving 2 separate prompting processes. The second prompt is self-augmented. SC [28] serves as another complement to the CoT algorithm by sampling a diverse set of reasoning paths and marginalizing out reasoning paths to aggregate final answers. However, the algorithm is still limited by the left-to-right reasoning mechanism of LLM. Least to Most Prompting (LtM) [29], also an advancement of the CoT algorithm, decomposes a problem into a set of subproblems built upon each other and inputs the solutions of the previous sub-problem into the prompt of the next sub-problem to gradually solve each sub-problem. Generated Knowledge Approach (GKA) [30] enables LLM to generate potentially useful information related to a given question before generating the response through 2 intermediate steps: knowledge generation and knowledge integration. Diverse Verifier on Reasoning Steps (DiVeRSe) [31], on the other hand, improves the reliability of LLM answers by generating multiple reasoning paths.

B. *The Application Of LLM In Code Related Tasks.*

Since the introduction of ChatGPT, LLM has been widely applied in various subfields of code security research. Zhang et al. [32] utilized ChatGPT to generate vulnerability exploitation code. Liu et al. [33] applied GPT to the task of vulnerability description mapping and evaluation tasks. They provided certain prompts to ChatGPT and extracted the required information from its responses using regular expressions. Zhang et al. [34] proposed STEAM, a framework for bug fixing using LLM to simulate programmers' behaviors. Kang et al. introduced the LIBRO [35] model for exploring bug reproduction tasks. Given a bug report, LIBRO first constructs a prompt to query an LLM and subsequently uses this prompt to generate an initial set of candidate tests by querying the LLM multiple times. LIBRO subsequently processes the tests to make them executable in the target program, identifies and manages tests that potentially reproduce the bug.

The aforementioned researches demonstrate that with appropriate algorithmic design, LLM is capable of handling various specific tasks in the field of code analysis.

C. Researches On Webshell Detection Techniques.

According to the timeline, we categorize the researches in the field of webshell detection into 3 stages: Start Stage, Initial Development Stage and In-depth Development Stage. In the Start Stage, research methods are simple and have numerous flaws and deficiencies, such as limited private datasets, unreasonable feature extraction methods, oversimplified classifier structure design [36], [37], etc. In the Initial Development Stage, relevant studies explore and make progress in various aspects of the detection process. However, these methods still have certain shortcomings in the fields of detection dataset, method universality, feature extraction [38]–[42], etc. Theoretical innovations remain relatively scarce. In the In-depth Development Stage, simple individual classifiers or machine learning algorithms become less common, and related researches have penetrated into the theoretical process level of modeling methods [43]–[45]. However, from an overall point of view, researches related to webshell detection techniques are still in their early stages, largely due to the slow progress of the attacker's research, and the lack of advanced webshell escape sample generation algorithms in the field.

VII. Conclusion

In this paper, we propose Hybrid Prompt, a webshell escape sample generation prompt algorithm that combines various prompt strategies such as ToT, Few-shot CoT, SC, etc. Hybrid Prompt combines structured webshell module and F_e chain, utilizes auxiliary methods to inspire LLMs to perform self-assessment and optimization, and demonstrates excellent performance on LLMs with strong code reasoning capabilities (GPT-3.5, GPT-4, Code-llama-34B), enabling the generation of high-quality webshell escape sample datasets. The Hybrid Prompt algorithm also exhibits strong scalability and generalization capability, allowing for the addition of more modules and corresponding F_e chains to update escape strategies and expand to more webshell languages. Our further work includes combining LLM fine-tuning techniques with the Hybrid Prompt algorithm to further enhance the code generation capability of LLM and designing more advanced information compression algorithms to enhance the quality of sample generation.

Acknowledgment

This work was supported by the National Key Research and Development Program of China Under Grant Nos (2022YFB3103403), the Fundamental Research Funds for the Central Universities Under Grant Nos (YCJJ20230464), and the National Natural Science Foundation of China Under Grant Nos (6217071437, 62072200, 62127808).

References

[1] O. Starov, J. Dahse, S. S. Ahmad, T. Holz, and N. Nikiforakis, "No honor among thieves: A large-scale analysis of malicious web shells," in *Proceedings of the 25th International Conference on World Wide Web*, 2016, pp. 1021–1032.

[2] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, and A. Armando, "Functionality-preserving black-box optimization of adversarial windows malware," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 3469–3478, 2021.

[3] P. Peng, L. Yang, L. Song, and G. Wang, "Opening the blackbox of virustotal: Analyzing online phishing scan engines," in *Proceedings of the Internet Measurement Conference*, 2019, pp. 478–485.

[4] A. Hannousse and S. Yahiouche, "Handling webshell attacks: A systematic mapping and survey," *Computers & Security*, vol. 108, p. 102366, 2021.

[5] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European signal processing conference (EUSIPCO)*. IEEE, 2018, pp. 533–537.

[6] W. Song, X. Li, S. Afroz, D. Garg, D. Kuznetsov, and H. Yin, "Mab-malware: A reinforcement learning framework for blackbox generation of adversarial malware," in *Proceedings of the 2022 ACM on Asia conference on computer and communications security*, 2022, pp. 990–1003.

[7] R. Labaca-Castro, B. Biggio, and G. Dreo Rodosek, "Poster: Attacking malware classifiers by crafting gradient-attacks that preserve functionality," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 2565–2567.

[8] Y. Chang, X. Wang, J. Wang, Y. Wu, K. Zhu, H. Chen, L. Yang, X. Yi, C. Wang, Y. Wang *et al.*, "A survey on evaluation of large language models," *arXiv preprint arXiv:2307.03109*, 2023.

[9] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[10] R. C. Staudemeyer and E. R. Morris, "Understanding lstm–a tutorial into long short-term memory recurrent neural networks," *arXiv preprint arXiv:1909.09586*, 2019.

[11] R. Dey and F. M. Salem, "Gate-variants of gated recurrent unit (gru) neural networks," in *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*. IEEE, 2017, pp. 1597–1600.

[12] C. Liu, X. Bao, H. Zhang, N. Zhang, H. Hu, X. Zhang, and M. Yan, "Improving chatgpt prompt for code generation," *arXiv preprint arXiv:2305.08360*, 2023.

[13] G. Deng, Y. Liu, V. Mayoral-Vilches, P. Liu, Y. Li, Y. Xu, T. Zhang, Y. Liu, M. Pinzger, and S. Rass, "Pentestgpt: An llm-empowered automatic penetration testing tool," *arXiv preprint arXiv:2308.06782*, 2023.

[14] Y. Sun, D. Wu, Y. Xue, H. Liu, H. Wang, Z. Xu, X. Xie, and Y. Liu, "When gpt meets program analysis: Towards intelligent detection of smart contract logic vulnerabilities in gptscan," *arXiv preprint arXiv:2308.03314*, 2023.

[15] Y. Wei, C. S. Xia, and L. Zhang, "Copiloting the copilots: Fusing large language models with completion engines for automated program repair," *arXiv preprint arXiv:2309.00608*, 2023.

[16] J. Zhao, Y. Rong, Y. Guo, Y. He, and H. Chen, "Understanding programs by exploiting (fuzzing) test cases," *arXiv preprint arXiv:2305.13592*, 2023.

[17] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining zero-shot vulnerability repair with large language models," in *2023 IEEE Symposium on Security and Privacy (SP)*, 2023, pp. 2339–2356.

[18] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh, "Autoprompt: Eliciting knowledge from language models with automatically generated prompts," *arXiv preprint arXiv:2010.15980*, 2020.

[19] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022.

[20] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of thoughts: Deliberate problem solving with large language models," *arXiv preprint arXiv:2305.10601*, 2023.

[21] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199–22 213, 2022.

[22] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, "Language models as knowledge bases?" *arXiv preprint arXiv:1909.01066*, 2019.

[23] X. L. Li and P. Liang, "Prefix-tuning: Optimizing continuous prompts for generation," *arXiv preprint arXiv:2101.00190*, 2021.

[24] B. Pang, G. Liang, J. Yang, Y. Chen, X. Wang, and W. He, "Cwsogg: Catching web shell obfuscation based on genetic algorithm and generative adversarial network," *The Computer Journal*, vol. 66, no. 5, pp. 1295–1309, 2023.

[25] Z. Du, Y. Qian, X. Liu, M. Ding, J. Qiu, Z. Yang, and J. Tang, "Glm: General language model pretraining with autoregressive blank infilling," *arXiv preprint arXiv:2103.10360*, 2021.

[26] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever, "Zero-shot text-to-image generation," in *International Conference on Machine Learning*. PMLR, 2021, pp. 8821–8831.

[27] W. Zhuang, C. Chen, and L. Lyu, "When foundation model meets federated learning: Motivations, challenges, and future directions," *arXiv preprint arXiv:2306.15546*, 2023.

[28] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[29] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. Le *et al.*, "Least-to-most prompting enables complex reasoning in large language models," *arXiv preprint arXiv:2205.10625*, 2022.

[30] J. Liu, A. Liu, X. Lu, S. Welleck, P. West, R. L. Bras, Y. Choi, and H. Hajishirzi, "Generated knowledge prompting for commonsense reasoning," *arXiv preprint arXiv:2110.08387*, 2021.

[31] Y. Li, Z. Lin, S. Zhang, Q. Fu, B. Chen, J.-G. Lou, and W. Chen, "Making language models better reasoners with step-aware verifier," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023, pp. 5315–5333.

[32] Y. Zhang, W. Song, Z. Ji, N. Meng *et al.*, "How well does llm generate security tests?" *arXiv preprint arXiv:2310.00710*, 2023.

[33] X. Liu, Y. Tan, Z. Xiao, J. Zhuge, and R. Zhou, "Not the end of story: An evaluation of chatgpt-driven vulnerability description mappings," in *Findings of the Association for Computational Linguistics: ACL 2023*, 2023, pp. 3724–3731.

[34] Y. Zhang, Z. Jin, Y. Xing, and G. Li, "Steam: Simulating the interactive behavior of programmers for automatic bug fixing," *arXiv preprint arXiv:2308.14460*, 2023.

[35] S. Kang, J. Yoon, and S. Yoo, "Large language models are few-shot testers: Exploring llm-based general bug reproduction," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 2312–2323.

[36] Y. Tian, J. Wang, Z. Zhou, and S. Zhou, "Cnn-webshell: malicious web shell detection with convolutional neural network," in *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, 2017, pp. 75–79.

[37] Z. Zhang, M. Li, L. Zhu, and X. Li, "Smartdetect: a smart detection scheme for malicious web shell codes via ensemble learning," in *Smart Computing and Communication: Third International Conference, SmartCom 2018, Tokyo, Japan, December 10–12, 2018, Proceedings 3*. Springer, 2018, pp. 196–205.

[38] Y. Wu, Y. Sun, C. Huang, P. Jia, and L. Liu, "Session-based webshell detection using machine learning in web logs," *Security and Communication Networks*, vol. 2019, pp. 1–11, 2019.

[39] L. Jinping, T. Zhi, M. Jian, G. Zhiling, and Z. Jiemin, "Mixed-models method based on machine learning in detecting webshell attack," in *Proceedings of the 2020 International Conference on Computers, Information Processing and Advanced Education*, 2020, pp. 251–259.

[40] H. Zhang, M. Liu, Z. Yue, Z. Xue, Y. Shi, and X. He, "A php and jsp web shell detection system with text processing based on machine learning," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*. IEEE, 2020, pp. 1584–1591.

[41] H. V. Le, T. N. Nguyen, H. N. Nguyen, and L. Le, "An efficient hybrid webshell detection method for webserver of marine transportation systems," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[42] Z. Zhou, L. Li, and X. Zhao, "Webshell detection technology based on deep learning," in *2021 7th IEEE Intl Conference on Big Data Security on Cloud (BigDataSecurity), IEEE Intl Conference on High Performance and Smart Computing,(HPSC) and IEEE Intl Conference on Intelligent Data and Security (IDS)*. IEEE, 2021, pp. 52–56.

[43] T. An, X. Shui, and H. Gao, "Deep learning based webshell detection coping with long text and lexical ambiguity," in *International Conference on Information and Communications Security*. Springer, 2022, pp. 438–457.

[44] B. Cheng, Y. Guo, Y. Ren, G. Yang, and G. Xu, "Msdetector: A static php webshell detection system based on deep-learning," in *International Symposium on Theoretical Aspects of Software Engineering*. Springer, 2022, pp. 155–172.

[45] A. Pu, X. Feng, Y. Zhang, X. Wan, J. Han, and C. Huang, "Bert-embedding-based jsp webshell detection on bytecode level using xgboost," *Security and Communication Networks*, vol. 2022, 2022.